Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Vložitev vozlišč omrežja v linearni prostorski zahtevnosti

Vid Kocijan

Delo je pripravljeno v skladu s Pravilnikom o podeljevanju
Prešernovih nagrad študentom,
pod mentorstvom prof. dr. Janeza Demšarja
in somentorstvom doc. dr. Jureta Leskovca

Ljubljana, 2017

Povzetek

Naslov: Vložitev vozlišč omrežja v linearni prostorski zahtevnosti

Avtor: Vid Kocijan

Da bi za napovedovanje obnašanja omrežij lahko uporabili algoritme za strojno učenje, moramo vozlišča omrežja predstaviti kot vektorje v nizkodimenzionalnem vektorskem prostoru. Trenutno najučinkovitejši algoritem za računanje vložitve vozlišč omrežja v vektorski prostor je Node2vec, ki omrežje vzorči s pristranskimi naključnimi sprehodi drugega reda. Algoritem Node2vec ima žal visoko pomnilniško zahtevnost zaradi velike količine predpomnjenih tabel verjetnostnih porazdelitev, kar povprečnemu uporabniku onemogoči uporabo na večjih omrežjih. V tem diplomskem delu je predstavljen hevristični pristop k simulaciji naključnih sprehodov z binarnimi drevesi, ki zagotavlja linearno časovno in pomnilniško zahtevnost simulacije, a hkrati ohranja kvaliteto izračunanih značilk. Hevristični pristop je na preizkušenih naborih podatkov porabil od 6-krat pa do 40-krat manj pomnilnika kot algoritem Node2vec.

Ključne besede: Vložitev vozlišč, omrežje, naključni sprehodi.

Abstract

Title: Vertex embeddings in linear space complexity

Author: Vid Kocijan

In order to predict the behaviour of networks with machine-learning algorithms, the vector representation of nodes in a low dimensional vector space is required. The current state-of-the-art algorithm for the calculation of node embeddings in vector space is Node2vec. Node2vec samples the network through the 2^{nd} order random walks. Unfortunately, Node2vec has a high memory complexity due to the preprocessed probability-distribution tables. Due to high memory complexity, an average user is unable to use it for larger networks. In this thesis, we present a heuristic approach to the random walk simulation. The heuristic approach replaces probability tables with binary trees and guarantees linear time and space complexity, while retaining the quality of computed features. The heuristic approach requires from 6 up to 40 times less memory than Node2vec on tested datasets.

Keywords: vertex embeddings, network, random walks.

Kazalo

Poglavje 1

Uvod

Napovedovanje obnašanja omrežij je pogosta naloga modernih aplikacij. Uporablja se za napovedovanje prijateljstev na socialnih omrežjih, za analizo cestnih omrežij, napovedovanje interakcij med geni, itd. Algoritmi ponavadi napovedujejo nove povezave ali pa oznake oz. lastnosti posameznih vozlišč. Za reševanje problemov, kjer iz statističnih podatkov napovedujemo nadaljnje obnašanje sistema, ponavadi uporabimo strojno učenje. Za to moramo vozlišča čim bolj reprezentativno predstaviti kot vektorje v nizkodimenzionalnem vektorskem prostoru.

Da tak pristop deluje, mora vektorska reprezentacija vozlišč omrežja čim bolje povzeti njihove lastnosti. Tako predstavitev pogosto ustvari strokovnjak s področja, s katerega prihaja problem. Ta pristop je počasen in reši le specifičen primer, prav tako pa pogosto ne pripelje do dobrih rezultatov, saj človek težko oceni, kakšne značilke dobro opišejo vlogo nekega vozlišča v omrežju. Bolj učinkovit pristop k problemu je, da programu pustimo, da se sam „nauči“ dobre vložitve.

Trenutno najučinkovitejši algoritem za ta namen je Node2vec, ki v preizkušenih naborih podatkov vse ostale znane pristope k problemu preseže v točnosti napovedi [5]. Žal algoritem Node2vec za svoje delovanje potrebuje veliko računalniškega pomnilnika, kar onemogoča njegovo uporabo na velikih omrežjih. V gostih omrežjih količina porabljenega pomnilnika namreč

narašča kvadratično. V tej diplomski nalogi je predstavljena izboljšava, ki z uporabo k-d dreves in različnih hevristik zagotovi, da poraba pomnilnika ostane linearna. Pristop je preizkušen na več naborih podatkov, kjer so rezultati hevrističnega pristopa enako kvalitetni kot rezultati algoritma Node2vec.

V drugem poglavju tega dela je opisano delovanje algoritma Node2vec in njegove pomanjkljivosti. V tretjem poglavju je opisan hevristični pristop k simuliranju naključnih sprehodov drugega reda ter paralelizacija simulacije naključnih sprehodov in računanja značilk. V četrtem poglavju je opisana eksperimentalna primerjava delovanja in rezultatov hevrističnega pristopa z delovanjem in rezultati algoritma Node2vec.

Poglavje 2

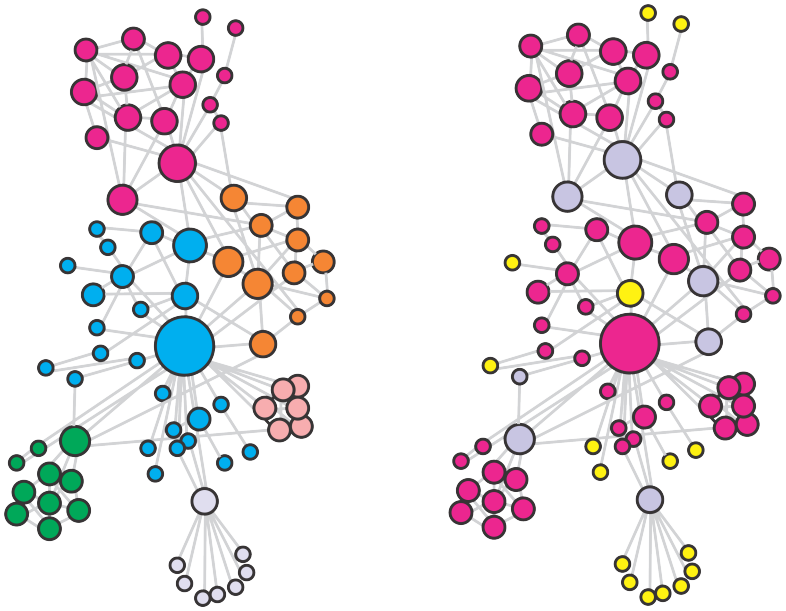
Računanje vložitev z algoritmom Node2vec

Algoritem Node2vec lahko v grobem razdelimo na dva dela. V prvem delu vzorčimo naključne sprehode po grafu in generiramo množico pristranskih naključnih sprehodov. V drugem delu pridobljene naključne sprehode uporabimo kot vhodne podatke za sestavljanje modela vektorjev. Največji izziv danega problema je določiti, kakšna je „dobra“ vektorska predstavitev vozlišč in sestaviti namensko funkcijo (angl. objective function), ki bo dobro opisala ta cilj, hkrati pa bo lahko izračunljiva.

Omrežje predstavimo kot utežen graf $G = (V, E, W)$. Če graf ni utežen, so vse uteži $w \in W$ enake 1. Naj bo $f : V \to \mathbb{R}^d$ preslikava vozlišč $v \in V$ v d-dimenzionalni vektorski prostor, ki jo želimo določiti, kjer je d dimenzija vektorskega prostora. Funkcijo f lahko predstavimo kot matriko velikosti $|V| \times d$. Modeliranje vektorske predstavitve vozlišč predstavimo kot optimizacijski problem z razširitvijo arhitekture Skip-Gram [7] na omrežja [9].

2.1 Okolica vozlišča

Naj bo $v \in V$ vozlišče. Definirajmo $N_S(v) \subset V$ kot okolico vozlišča v, dobljeno s strategijo generiranja pristranskih naključnih sprehodov S. $N_S(v)$



Slika 2.1: Sopojavitev likov romana Nesrečniki. Vozlišča s podobno vektorsko predstavitvijo so pobarvana z isto barvo. Barve vozlišč prikazujejo homofilijo (levo) in strukturno ekvivalenco (desno), kot ju je določil algoritem Node2vec. [5]

predstavlja množico vozlišč, ki so v omrežju podobna vozlišču v. Torej okolica vozlišča v ni nujno množica vozlišč v njegovi bližini, pač pa množica vozlišč, za katera želimo, da imajo podobno vektorsko predstavitev kot v. Relacija „biti v okolici“ $u \in N_S(v)$ je simetrična. Okolica vozlišča se lahko razlikuje glede na uporabljeno strategijo S. Med strategijami ločimo ekstrema: vozlišči sta si lahko podobni, ker se nahajata eno blizu drugemu (homofilija) ali pa ker imata podobno vlogo v omrežju (strukturna ekvivalenca), kot je razvidno iz slike 2.1. Na levi strani barve razdelijo vozlišča v omrežju glede na njihovo lokacijo, na desni strani pa glede na njihovo vlogo. Množice vozlišč, kot jih v desnem primeru ustvari Node2vec bi lahko grobo opisali kot „stranska vozlišča“ rumene barve, „mostove“ sive barve in „preostala vozlišča“ rožnate barve.

2.1.1 Modeliranje okolice s pristranskimi naključnimi sprehodi 2. reda

Okolico vozlišča v določimo s simulacijo več pristranskih naključnih sprehodov fiksne dolžine l. Če sta dve vozlišči v sprehodu oddaljeni za manj kot k korakov, se nahajata v okolici drug drugega. Bolj formalno, naj bo $c = (c_1, c_2, \ldots, c_l); c_i \in V$ sprehod dolžine l, simuliran s strategijo S. Okolica vozlišča c_j je $N_s(c_j) = \{c_i : j - k \leq i \leq j + k\}$

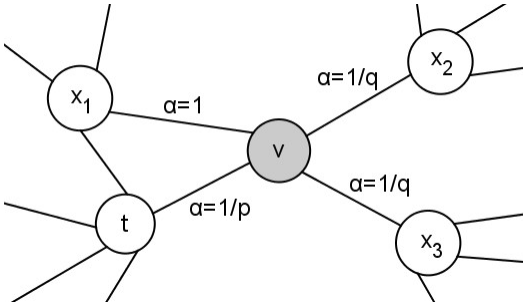
V vsakem koraku naslednje vozlišče izberemo po sledečem postopku. Naj bo c_i i-to, še ne določeno vozlišče v sprehodu in $c_{i-1} = v$. Verjetnost, da v naslednjem koraku obiščemo vozlišče x je sledeča:

$$P(c_i = x | c_{i-1} = v) = \begin{cases} \frac{\pi_{v,x}}{Z}, & \text{če } (v,x) \in E \\ 0, & \text{sicer} \end{cases}$$

$\pi_{v,x}$ je nenormalizirana verjetnost premika $v \to x$, Z pa je normalizacijska konstanta $Z = \sum_{y \in V} \pi_{v,y}$. Nenormalizirana verjetnost premika $\pi_{v,x} = w_{v,x} \cdot \alpha_{p,q}(v,x)$, kjer je $w_{v,x}$ teža povezave $e_{v,x}$, $\alpha_{p,q}(v,x)$ pa funkcija pristranskosti. Funkcija pristranskosti je določena s strategijo S, ki jo opišemo z vhodnima parametroma p in q. Naj bo $t = c_{i-2}$ predzadnje obiskano vozlišče in $d_{t,x} \in \{0,1,2\}$ razdalja med vozliščema t in x. Potem bo funkcija pristranskosti sledeča:

$$\alpha_{p,q}(t,x) = \begin{cases} \frac{1}{p}, & \text{če } t = x \\ 1, & \text{če } d_{t,x} = 1 \\ \frac{1}{q}, & \text{če } d_{t,x} = 2 \end{cases}$$

S parametroma p in q določimo verjetnost izbire posameznega vozlišča glede na to, kateri dve vozlišči smo obiskali zadnji. Zato pravimo, da je to naključni sprehod drugega reda. Ker ima $d_{t,x}$ največ tri možne vrednosti, več kot dveh parametrov za opis strategije ne potrebujemo. Parameter p določa, kako verjetno se bomo vrnili na vozlišče, ki smo ga pravkar obiskali, q pa, kako verjetno obiščemo vozlišče, ki ni sosednje predzadnjemu vozlišču, kot prikazano na sliki 2.2.



Slika 2.2: Ilustracija naključnega sprehoda. Oznake na povezavah prikazujejo funkcijo α. [5]

Parameter vračanja, p (angl. Return parameter): Vrednost parametra p določa verjetnost vrnitve na pravkar obiskano vozlišče. Če ga nastavimo na visoko vrednost, višjo od $\max(q, 1)$, zmanjšamo verjetnost vračanja na pravkar obiskana vozlišča. S tem zmanjšamo verjetnost ponavljujočih zaporedij s periodo 2. Po drugi strani pa lahko z majhno vrednostjo, manjšo od $\min(q, 1)$ povečamo vračanje na pravkar obiskano vozlišče in s tem pregledovanje lokalnega območja v omrežju.

Parameter izhoda, q (angl. In-Out parameter): Vrednost parametra q določa verjetnost obiska vozlišča, ki ni sosednje predzadnjemu obiskanemu vozlišču. Izhodne povezave razdeli na dve podmnožici, „zunanje“ in „notranje“. Na sliki 2.2 sta izhodni povezavi (v, x_3) in (v, x_2) „zunanji“, povezava (v, x_1) pa „notranja“. Če parameter q nastavimo na visoko vrednost (višjo od $\max(p, 1)$), zmanjšamo verjetnost obiskovanja zunanje okolice in prehoda mostov. Naključni sprehod se bo obnašal podobno kot preiskovanje v širino in vzorčil predvsem relacije med vozlišči blizu skupaj. Če parameter q nastavimo na nizko vrednost (nižjo od $\min(p, q)$), povečamo verjetnost izhoda iz „lokalnega“ podgrafa. Naključni sprehod tako postane bolj podoben preiskovanju v globino in bo z večjo verjetnostjo obiskal vozlišča, ki niso del „lokalnega“ podgrafa.

2.2 Namenska funkcija in model Skip-Gram

Namenska funkcija je funkcija, ki slika značilke vektorjev v realna števila in opiše kvaliteto značilk. To pomeni, da ima visoko vrednost, če vektorska predstavitev dobro odraža relacije med vozlišči in nizko sicer.

Skip-Gram je jezikovni model prvotno namenjen modeliranju naravnega jezika [7], ki maksimizira verjetnost sopojavitve besed, ki se v stavkih nahajajo ena blizu druge. Z vpeljavo okolice 2.1 to definicijo razširimo na omrežja in maksimiziramo logaritem verjetnosti, da opazujemo element okolice vozlišča u glede na vektorsko predstavitev vozlišča u. Naš cilj je, da se $f(u)$ čim bolj ujema z vektorskimi predstavitvami vozlišč iz svoje okolice. Po Skip-gram arhitekturi takih sistemov to pomeni, da želimo iz vrednosti $f(u)$ čim bolje napovedati, katera druga vozlišča se nahajajo v okolici $N_S(u)$ (za razliko od arhitekture „Continuous bag of words“, kjer iz opisa okolice napovedujemo vozlišče u). To lahko izrazimo z namensko funkcijo:

$$\max_f \sum_{u \in V} \log Pr(N_S(u)|f(u))$$

Da je zgornja enačba izračunljiva, privzamemo nekaj poenostavitev:

- Pogojna neodvisnost elementov okolice. Predpostavimo, da so vozlišča v okolici $n_i \in N_S(u)$ neodvisna od drugih vozlišč v okolici. Verjetnost, da pravilno napovemo okolico nekega vozlišča tako poenostavimo na produkt verjetnosti, da pravilno napovemo njene elemente.

$$Pr(N_S(u)|f(u)) = \prod_{n_i \in N_S(u)} Pr(n_i|f(u))$$

- Simetrija okolice. Poleg simetričnosti okolice, kot definirano v razdelku 2.1, predpostavimo še, da je simetrična tudi verjetnost. Torej če velja $u \in N_s(v)$, sledi $P(u|f(v)) = P(v|f(u))$.

- Verjetnostno porazdelitev vozlišča po okolicah lahko modeliramo kot enoto „softmax“ (angl. softmax unit).

$$Pr(n_i|f(u)) = \frac{\exp(f(n_i)^T f(u))}{\sum_{v \in V} \exp(f(v)^T f(u))}$$

S temi poenostavitvami lahko prejšnjo namensko funkcijo preoblikujemo:

$$\max_f \sum_{u \in V} \left[-\log Z_u + \sum_{n_i \in N_u} f(n_i)^T f(u) \right]$$

Z_u je imenovalec enote „softmax“ $Z_u = \sum_{v \in V} \exp(f(u)^T f(v))$.

2.3 Simulacija naključnih sprehodov

Kot definirano v razdelku 2.1.1, lahko izračun okolice vozlišča modeliramo z množico naključnih sprehodov. Omrežje vzorčimo z $r \cdot |V|$ sprehodi dolžine l tako, da v vsakem vozlišču začnemo r sprehodov. Na naborih podatkov, na katerih je bil algoritem testiran, so se dobro obnesle vrednosti $l = 80$ in $r = 10$. Za učinkovito izbiranje naslednjega vozlišča vnaprej izračunamo verjetnosti prehoda med vsakim možnim parom vozlišč. Ker je ta verjetnost odvisna od predzadnjega obiskanega vozlišča v sprehodu, ima izračun vseh verjetnosti prehoda pomnilniško zahtevnost $O(d^2|V|)$, kjer je d povprečna stopnja vozlišča. V resničnih omrežjih je d relativno nizek. Naslednje vozlišče pri dani diskretni verjetnostni porazdelitvi lahko izberemo v konstantnem času z $O(n)$ predpriprave z metodo „Alias sampling“ [10], kjer lahko z metom pravične kocke in metom pristranskega kovanca izberemo naključen element množice glede na vnaprej podano diskretno verjetnostno porazdelitev. Ko simuliramo naključni sprehod dolžine $l > k$, ustvarimo k vzorcev okolice za $l - k$ vozlišč. Ker iz enega sprehoda dolžine l ustvarimo $k(l - k)$ vzorcev to vpelje neželjeno pristranskost vhodnih podatkov, saj vzorci niso popolnoma neodvisni eden od drugega. Ustvarjeno pristranskost zaradi simetrije okolice zanemarimo, s tem pa zelo zvišamo učinkovitost vzorčenja. Torej za simuliranje naključnih sprehodov porabimo $O(d^2|V|)$ časa in pomnilnika za predpripravo in $O(l \cdot r \cdot |V|)$ časa in pomnilnika za simuliranje sprehodov. Predpriprava in simuliranje sprehodov sta koraka, ki ju izvedemo zaporedno, vendar sta vsak zase lahko paralelizabilna.

Algoritem 1: Algoritem Node2vec

Data: Graf $G = (V, E, W)$, dimenzija d, število sprehodov iz vozlišča r, dolžina sprehoda l, širina okolice k, Parametra p in q
Result: $f : \mathbb{R}^{|V| \times d}$
π = PredpripravaVerjetnostnihPorazdelitev(G, p, q);
Sprehodi = [];
for $Iter \leftarrow 1$ **to** r **do**
  for $v \in V$ **do**
    sprehod = [v];
    for *walk_iter* $\leftarrow 1$ **to** l **do**
      t_{-2}, t_{-1} =sprehod[-2],sprehod[-1];
      s =AliasSample($t_{-1}, \pi_{t_{-2}}$);
      Dodaj s k sprehod;
    end
    Dodaj sprehod k Sprehodi;
  end
end
f = IzračunZnačilk(k,d,Sprehodi);

2.4 Izračun značilk

Maksimum namenske funkcije iz razdelka 2.2 iščemo z algoritmom SGD (Stochastic Gradient Descent). Začetna vrednost matrike f je naključna. V vsaki iteraciji vsebini f prištejemo gradient (smer najstrmejšega vzpona) pomnožen s hitrostjo učenja α. Hitrost učenja α ima začetno vrednost tipično 0.025, tekom algoritma pa jo postopoma enakomerno zmanjšujemo do 0, da vrednost f skonvergira. Algoritem 2 za izračun funkcije f je neučinkovit, saj v vsaki iteraciji spreminja celotno matriko f. Da ga pospešimo, privzamemo nekaj poenostavitev.

Algoritem 2: Funkcija IzračunZnačilk

Data: širina okolice k, dimenzija d, množica sprehodov Sprehodi
Result: $f : \mathbb{R}^{|V| \times d}$
$f = \mathbb{R}^{|V| \times d}$ naključnih vrednosti;
for *sprehod* $\in$ *Sprehodi* **do**
  for $v_j \in$ *Sprehod* **do**
    for $u \in$*Sprehod*$[j-k : j+k]$ **do**
      $J(f) = -\log P(u|f(v_j))$;
      $f = f - \alpha \cdot \frac{\partial J}{\partial f}$;
    end
  end
end

Pri izračunu $\log P(u|f(v_j))$ in $\frac{\partial J}{\partial f}$ zanemarimo spremembe $f(w)$ če $w \neq v_j$ in $w \neq u$. Torej $\frac{\partial J}{\partial f}(w) = 0 \Leftrightarrow w \neq v_j \wedge w \neq u$. Ker je izračun vrednosti Z_u računsko zahteven, ga nadomestimo s približkom. Najbolj učinkovit postopek za izračun približka je negativno vzorčenje [8]. Vrednost Z_u ocenimo z vrednostmi m naključno izbranih vektorjev iz matrike f. Za reprezentativno izbiro naključnih vektorjev potrebujemo frekvenco pojavitev nekega vozlišča v sprehodih. Večje je omrežje, manjši m zadošča za reprezentativen približek. Za izračun Z_u node2vec tipično uporabi $m = 5$, torej lahko vrednost

Z_u izračunamo v času $O(1)$. Ker vrstni red sprehodov ni pomemben, lahko tudi izračun značilk izvajamo paralelno na več procesorjih.

2.5 Pomanjkljivosti algoritma Node2vec

Kljub dobrim rezultatom [5], algoritem Node2vec porabi veliko količino pomnilnika. Pomnilniška in časovna zahtevnost sprehodov in predpriprave 2.3 nanj je namreč linearna zgolj pri predpostavki, da je povprečna stopnja vozlišča d majhna in da je stopnja vozlišča porazdeljena normalno ali enakomerno. Večina realnih omrežij ima stopnjo vozlišča porazdeljeno eksponentno [9]. V primeru socialnih omrežjih ima majhna podmnožica posameznikov (zvezdnikov) veliko več sledilcev, kot povprečen uporabnik. Podobno ima tudi na svetovnem spletu majhna podmnožica spletnih strani veliko več obiskov ali vhodnih povezv kot povprečna spletna stran. Časovna in pomnilniška zahtevnost predpriprave, ki je enaka $\sum_{v \in V} indeg(v) \cdot outdeg(v)$, tako kljub nizki povprečni stopnji vozlišča narašča kvadratično.

Drugi vzrok visoke porabe pomnilnika je shranjevanje vseh simuliranih naključnih sprehodov. Kot je razvidno iz algoritma 1, se vsak simulirani sprehod shrani v pomnilnik. Poleg uporabe vzorcev pri računanju značilk, kot razvidno iz algoritma 2, to tabelo uporabimo za računanje frekvence pojavitve nekega vozlišča v sprehodu, ki jo potrebujemo za negativno vzorčenje [8]. Shranjevanje simuliranih sprehodov prav tako omogoča večkratno iteracijo čez podatke. Ta tabela ima $|V|r \times l$ elementov.

Poraba pomnilnika algoritma Node2vec je bila testirana na implementaciji v programskem jeziku C++, dostopni kot del Stanford Network Analysis Platform [6]. Program je na naboru podatkov BlogCatalog [11] (10.312 vozlišč, 333.983 povezav) porabil $4,5GB$ pomnilnika, na naboru podatkov LiveJournal[1] (4.847.571 vozlišč, 68.993.773 povezav) pa $210GB$ pomnilnika. Iz izmerjenih podatkov je očitno, da poraba pomnilnika že pri relativno majhnih omrežjih preseže zmogljivosti naprav, ki so dostopne povprečnemu uporabniku.

Poglavje 3

Hevristični pristop k pristranskim naključnim sprehodom 2. reda

Za zmanjšanje pomnilniške zahtevnosti simuliranja pristranskih naključnih sprehodov se moramo znebiti predpripravljenih verjetnostnih porazdelitev za vsak možen par zadnjih dveh obiskanih vozlišč v omrežju. Ker je število korakov v simulaciji naključnih sprehodov relativno veliko ($|V| \cdot r \cdot l$), mora izbira naslednjega vozlišča v sprehodu še vedno potekati v času $O(1)$. V hevurističnem pristopu, opisanem v sledečih razdelkih, verjetnostne tabele nadomestimo z binarnimi drevesi, ki so neodvisna od predzadnjega obiskanega vozlišča.

3.1 Parametrizacija vozlišč z vektorji oddaljenosti

Predpostavimo, da so povezave grafa G neuteženе in neusmerjene. Naključno izberimo M različnih vozlišč $m_1, \ldots m_M \in V$. V praksi se je primerno obnesla vrednost $M = 5$. Vsakemu vozlišču $v \in V$ dodelimo M-dimenzionalen vektor razdalj do izbranih vozlišč $r_v = (d_{m_1,v}, \ldots d_{m_M,v})$. Naj bosta v in x sosednji

vozlišči grafa G. Očitno velja $r_v - r_x \in \{-1, 0, 1\}^M$. Če za poljubni vozlišči $y, z \in V$ velja $r_y - r_z \notin \{-1, 0, 1\}^M$, vemo, da y in z nista sosednji.

3.2 Nadomestitev tabel verjetnostnih porazdelitev z vektorji oddaljenosti

Za vsako vozlišče v izračunamo $r_v - r_x$ za vsako vozlišče x, sosednje v. Fiksirajmo vozlišče v, ki je sosednje vozliščem $x_1, \ldots x_d$, kjer je d izhodna stopnja vozlišča v. Izhodne povezave, ki vodijo iz vozlišča v so povezave $e_{v,x_1}, \ldots e_{v,x_d}$. Omejimo se zgolj na omenjene povezave. Označimo teže izhodnih povezav z $w_i = w(e_{v,x_i})$ in vektorske predstavitve teh povezav z $q_i = r_v - r_{x_i}$.

3.2.1 Gradnja k-d drevesa

Vektorje $q_1 \ldots q_d$ uredimo v k-d drevo T [2]. Izbiranje naslednjega vozlišča v naključnem sprehodu je očitno ekvivalentno izbiri naključnega vektorja iz T. Izbira naključnega vektorja iz T je naključni sprehod iz korena drevesa do nekega lista. Ker vemo, da so vektorji $q_1 \ldots q_d \in \{-1, 0, 1\}^M$ in ker izbiramo naključni element, lahko strukturo drevesa in njegovo gradnjo poenostavimo. Drevo T bo imelo globino največ M, po vsaki dimenziji elementov bomo sortirali največ enkrat.

Na neki globini izgradnje drevesa $i < M$ gradimo poddrevo T_a in vektorje delimo po i-ti dimenziji. Standardni algoritem za iskanje mediane [4], ki deluje v linearnem času, lahko nadomestimo s štetjem, kolikokrat se pojavi vrednost -1, kolikokrat 0 in kolikokrat 1. Rekurzivno zgradimo poddrevesi T_n in T_p. T_n je poddrevo z vektorji, ki so imeli v i-ti dimenziji vrednost manjšo ali enako od mediane („negativno“ poddrevo), T_p pa je poddrevo z vektorji, ki so imeli v i-ti dimenziji vrednost večjo ali enako od mediane („pozitivno“ poddrevo). Naj bo teža drevesa $w_T = \sum_{q_i \in T} w_i$. Rekurzivno lahko izračunamo $w_{T_a} = w_{T_p} + w_{T_n}$.

Za razliko od klasične definicije k-d drevesa lahko listi drevesa T še vedno

vsebujejo več vektorjev. Predpostavimo, da so si vektorji v listih že dovolj podobni med sabo, da množice vektorjev v listu nima smisla deliti še naprej. V vsak list L drevesa T zato vstavimo tabelo verjetnostne porazdelitve, tako da je vsak vektor $q_i \in L$ izbran z verjetnostjo $\frac{w_i}{w_L}$, kjer je $w_L = \sum_{q_i \in L} w_i$.

Tako binarno drevo po istem postopku zgradimo za vsako vozlišče $v \in V$.

3.2.2 Izbira naslednjega vozlišča v naključnem sprehodu

Naj bo t predzadnje in v zadnje vozlišče v naključnem sprehodu. T naj bo k-d drevo v vozlišču v, kot opisano v prejšnjem razdelku. Izbiro naslednjega vozlišča v sprehodu razdelimo v dve fazi. V prvi fazi naključno izberemo, če se vrnemo v vozlišče t. To storimo tako, da vržemo pristranski kovanec. Z verjetnostjo $\frac{\frac{w_{t,v}}{p}}{w_T - w_{t,v} + \frac{w_{t,v}}{p}} = \frac{w_{t,v}}{p \cdot w_T - w_{t,v} \cdot (p-1)}$ sprehod nadaljujemo v vozlišču t, sicer nadaljujemo z drugo fazo. Kot razvidno iz formule, je verjetnost vrnitve v t odvisna od razmerja med težo drevesa w_t in težo povezave $w_{t,v}$, skalirane z $\frac{1}{p}$.

Če v prvi fazi nismo izbrali vrnitve v vozlišče t, v drugi fazi izberemo eno iz izhodnih povezav, kar je ekvivalentno izbiri enega od listov v drevesu T. Izračunamo $\rho = r_v - r_t$, vektorsko predstavitev vozlišča t, s katerega smo pravkar prišli. Iz korena naredimo naključni sprehod po drevesu T, tako da v vsakem vozlišču vržemo pristranski kovanec. Recimo, da se nahajamo v korenu poddrevesa T_a, kot smo ga ustvarili v prejšnjem razdelku. Z verjetnostjo p_a bomo izbiranje naslednjega vozlišča v sprehodu nadaljevali v „pozitivnem“ poddrevesu, z verjetnostjo $1 - p_a$ pa v „negativnem“.

$$p_a = \begin{cases} \frac{w_{T_p}}{w_{T_p} + \frac{w_{T_n}}{q}}, & \text{če } \rho_i = 1 \\ \frac{w_{T_p}}{w_{T_p} + w_{T_n}}, & \text{če } \rho_i = 0 \\ \frac{w_{T_p}}{w_{T_p} + q \cdot w_{T_n}}, & \text{če } \rho_i = -1 \end{cases}$$

Če $\rho_i = 0$ o relaciji med vozliščema t in x ne moremo povedati dosti, torej pozitivno poddrevo izbiramo glede na razmerje med težama pozitivnega

in negativnega poddrevesa w_{T_p} in w_{T_n}. Če $\rho_i = -1$ vemo, da velja $q_{u_i} = 1 \implies d(t,u) = 2$. Taka vozlišča u se nahajajo v pozitivnem poddrevesu, zato težo pozitivnega poddrevesa dodatno množimo z $\frac{1}{q}$ in verjetnost obiska pozitivnega poddrevesa določimo glede na razmerje med w_{T_n} in $\frac{w_{T_p}}{q}$. Če $\rho_i = 1$ je izpeljava simetrična kot za $\rho_i = -1$.

Ko sprehod po drevesu zaključimo v enem od listov, izberemo naključnega od vektorjev, ki se nahajajo v njem, z verjetnostno porazdelitvijo, kot izračunano v prejšnjem razdelku.

Namesto predpomnjenja verjetnostne porazdelitve za vsako možno predzadnje vozlišče v naključnem sprehodu smo zgradili binarno drevo neodvisno od predzadnjega vozlišča v sprehodu. Predzadnje vozlišče v sprehodu je vplivalo le na sprehod skozi to drevo. Parametra p in q imata v hevristiki enaki vlogi v simulaciji naključnih sprehodov kot v algoritmu Node2vec. Kljub temu pri enakih vrednostih vhodnih parametrov p in q Node2vec in hevristični pristop ne bosta vrnila enakih rezultatov. Zato so optimalne vrednosti parametrov p in q za analizo nekega omrežja s hevrističnim pristopom morda drugačne od optimalnih vrednosti p in q za analizo istega omrežja z algoritmom Node2vec.

3.3 Analiza časovne in prostorske zahtevnosti hevrističnega pristopa

Velikost binarnega drevesa T v vozlišču v je $O(2^M + deg(v))$. Ker je M tipično nizka konstanta, npr. $M = 5$, jo lahko zanemarimo in velikost drevesa ocenimo na $O(deg(v))$. Skupna pomnilniška in časovna zahtevnost predpomnjenja je torej $O(\sum_{v \in V} deg(v))$, kar je po lemi o rokovanju enako $O(|E|)$.

Izbira naslednjega vozlišča v naključnem sprehodu ima časovno zahtevnost $O(M)$. V prvi fazi naredimo zgolj en met kovanca, kar lahko naredimo v konstantnem času. V drugi fazi prehodimo binarno drevo od korena do nekega lista. Drevo ima globino M, na vsakem nivoju vržemo kovanec, torej naredimo $O(M)$ korakov. Naključni vektor v listu drevesa lahko izberemo v

$O(1)$ z uporabo metode „alias sampling“ [10].

Ker v praksi $M = 5$ zadošča, smo izpolnili zahtevo o izbiri naslednjega vozlišča v konstantnem času.

3.4 Paralelizacija vzorčenja in računanja značilk

Kot omenjeno v razdelku 2.5, shranjevanje simuliranih naključnih sprehodov za kasnejšo analizo zavzame precej pomnilnika. Shranjevanju vseh sprehodov se lahko ognemo, če simuliranja sprehodov in računanja značilk ne izvajamo enega za drugim, ampak vzporedno. Ker je pri računanju značilk vsak sprehod samostojna enota, lahko sprehode generiramo sproti med računanjem značilk.

S tem izgubimo možnost večkratne iteracije čez generirane sprehode. Ker lahko sprehode generiramo precej hitreje, kot pa računamo značilke, lahko namesto, da večkrat uporabimo iste sprehode, preprosto generiramo več novih. Večkratna iteracija čez podatke se tipično uporablja kot nadomestitev za njihovo majhno količino, v tem algoritmu pa lahko sprehodov generiramo poljubno mnogo.

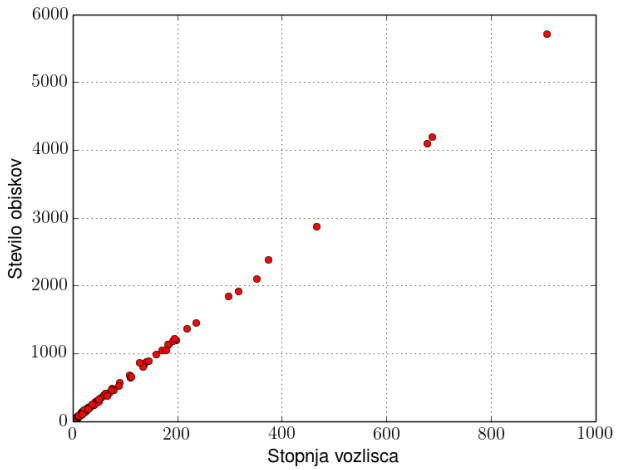
Druga ovira pri vzporednem izvajanju simulacije sprehodov in računanja značilk so frekvence pojavitev posameznega vozlišča, ki jih potrebujemo za negativno vzorčenje. Te nadomestimo s približkom.

3.5 Aproksimacija frekvence obiska s stopnjami vozlišč

Število obiskov vozlišča je linearno odvisno od stopnje vozlišča. Kot je vidno iz grafov 3.1 in 3.2, je to neodvisno od vhodnih parametrov.

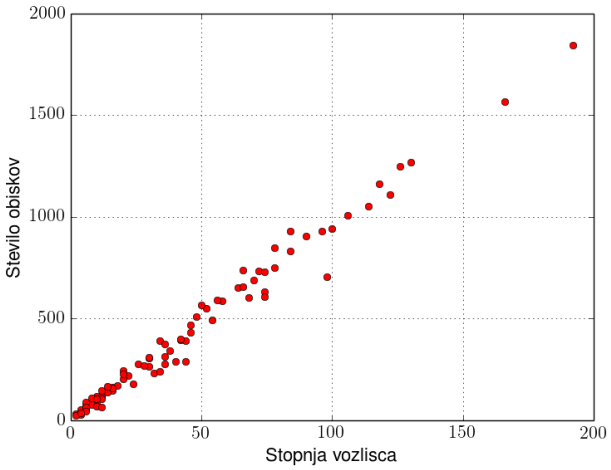
Ker sta število obiskov nekega vozlišča in njegova stopnja linearno odvisni, lahko frekvenco obiska nekega vozlišča ocenimo z njegovo stopnjo. Če število obiskov vozlišča v označimo s s_v, lahko frekvenco obiska izrazimo po naslednji formuli:

$$\frac{s_v}{|V| \cdot r \cdot l} \approx \frac{deg(v)}{\sum_{v \in V} deg(v)} = \frac{deg(v)}{2 \cdot |E|}$$



Slika 3.1: Relacija med stopnjo vozlišča in številom obiskov na 100 naključnih vozliščih na podatkovnem naboru BlogCatalog. Uporabljeni parametri $p = 1$, $q = 1$, $r = 10$ in $l = 80$

.



Slika 3.2: Relacija med stopnjo vozlišča in številom obiskov na 100 naključnih vozliščih na podatkovnem naboru LiveJournal. Uporabljeni parametri $p = 0.25$, $q = 4$, $r = 10$ in $l = 120$

Poglavje 4

Primerjava z algoritmom Node2vec

Pri primerjanju hevrističnega pristopa z algoritmom Node2vec nas zanimata:

- vpliv hevrističnega pristopa na kvaliteto rezultata,
- vpliv hevrističnega pristopa na časovno in pomnilniško zahtevnost.

Zaradi sprememb v algoritmu predpostavljamo, da se bo poraba pomnilnika opazno zmanjšala, kvaliteta rezultata in poraba časa pa bosta ostali približno enaki. Kljub temu, da se je z uporabo hevristik zmanjšala tudi časovna zahtevnost predpriprave, je že iz implementacije Node2vec algoritma razvidno, da je v praksi računanje značilk časovno bolj zahtevno kot simuliranje naključnih sprehodov.

4.1 Primerjava rezultatov

Za ocenjevanje kvalitete značilk je bila uporabljena večznačna klasifikacija (angl. multi-label classification) – vsako vozlišče je označeno z eno ali več značkami iz končne množice. Za učenje modela je bil uporabljen „eden proti vsem“ (angl. one-vs-rest) klasifikator za logistično regresijo z L2 regularizacijo. Za učenje je bil uporabljen nek vnaprej določen delež vozlišč in njihovih

značk, preostalim vozliščem pa so bile na podlagi naučenega modela napovedane značke. Točnost napovedanih podatkov je bila izmerjena z merami Mikro-F_1 in Makro-F_1 (angl. Micro-F_1 and Macro-F_1 scores).

Uporabljena sta bila sledeča nabora podatkov:

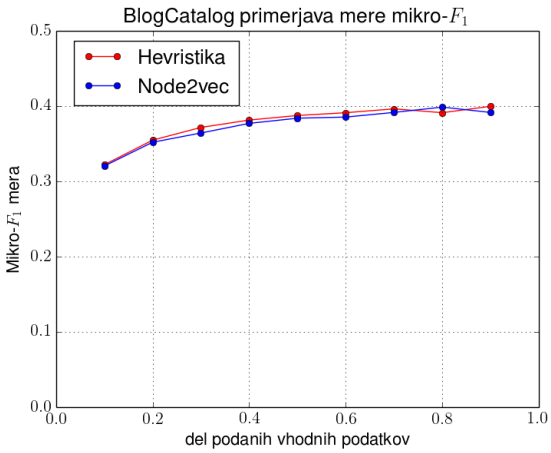
- BlogCatalog [11]: Omrežje družbenih relacij med blogerji na spletni strani BlogCatalog. Značke so različna zanimanja, ki so jih blogerji našteli sami skozi metapodatke in oznake svojih objav. Omrežje ima 10.312 vozlišč, 333.985 povezav in 39 različnih značk.

- Interakcije proteinov v človeškem telesu (v nadaljevanju PPI – Protein Protein Interaction) [3]: Podatkovni nabor je podgraf celotnega omrežja interakcij proteinov v človeškem telesu. Omrežje razpenjajo vozlišča, ki predstavljajo dobro raziskane in klasificirane proteine. Značke predstavljajo različna biološka stanja. Omrežje ima 3.890 vozlišč, 76.584 povezav in 50 različnih značk.

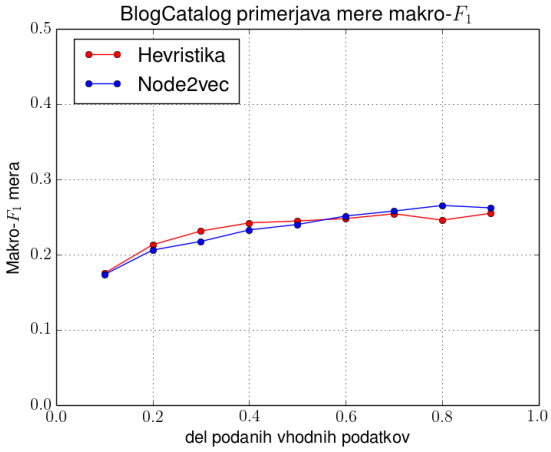
Nabora podatkov predstavljata omrežji z različnimi lastnostmi. V socialnem omrežju BlogCatalog so tipično povezani blogerji s podobnimi interesi, značke pa posledično predstavljajo homofilijo. V naboru podatkov PPI pričakujemo več strukturne ekvivalence med proteini, saj so v stiku predvsem proteini s komplementarno vlogo.

Optimalni parametri p in q so bili poiskani s preiskovanjem čez vse možne vrednosti para $(p, q) \in \{\frac{1}{4}, \frac{1}{2}, 1, \frac{3}{2}, 2, 4\}^2$. Za vsako možno vrednost vhodnih parametrov so bile 10-krat izračunane značilke in določena točnost napovedi pri $\frac{1}{2}$ podanih učnih podatkov. Za optimalne parametre so bili vzeti tisti, ki so imeli najvišje povprečje mere Mikro-F_1.

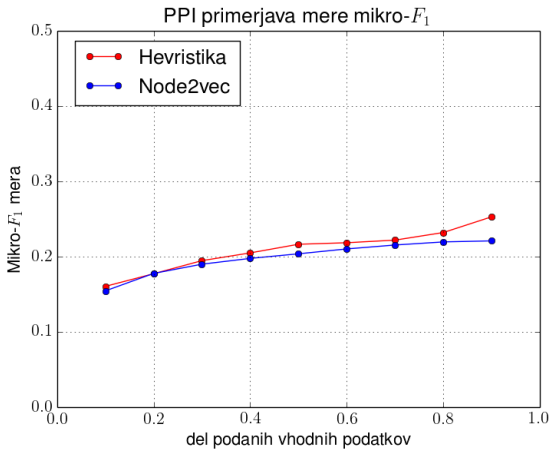
Iz grafov 4.1, 4.2, 4.3 in 4.4 je razvidno, da hevristični pristop izračuna primerljivo kvalitetne značilke kot algoritem Node2vec.



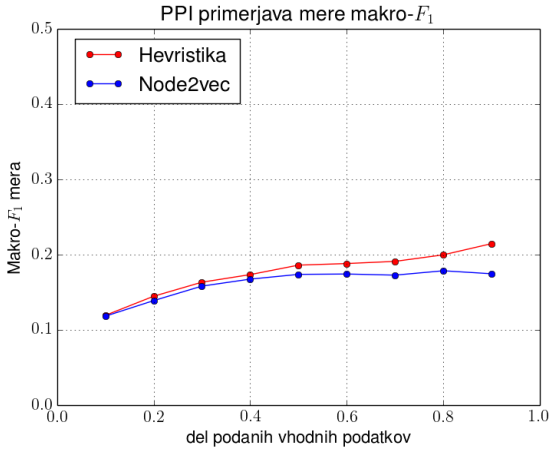
Slika 4.1: Primerjava rezultatov na naboru podatkov BlogCatalog z mero mikro-F_1 pri različnih deležih podatkov za učenje. Za algoritem Node2vec so bili uporabljeni parametri $p = \frac{1}{4}$ in $q = \frac{1}{4}$. Za hevristični pristop so bili uporabljeni parametri $p = 4$ in $q = \frac{3}{2}$.



Slika 4.2: Primerjava rezultatov na naboru podatkov BlogCatalog z mero makro-F_1 pri različnih deležih podatkov za učenje. Za algoritem Node2vec so bili uporabljeni parametri $p = \frac{1}{4}$ in $q = \frac{1}{4}$. Za hevristični pristop so bili uporabljeni parametri $p = 4$ in $q = \frac{3}{2}$.



Slika 4.3: Primerjava rezultatov na naboru podatkov PPI z mero mikro-F_1 pri različnih deležih podatkov za učenje. Za algoritem Node2vec so bili uporabljeni parametri $p = 4$ in $q = 1$. Za hevristični pristop so bili uporabljeni parametri $p = 1$ in $q = 1$.



Slika 4.4: Primerjava rezultatov na naboru podatkov PPI z mero makro-F_1 pri različnih deležih podatkov za učenje. Za algoritem Node2vec so bili uporabljeni parametri $p = 4$ in $q = 1$. Za hevristični pristop so bili uporabljeni parametri $p = 1$ in $q = 1$.

merjeni program	porabljen pomnilnik	porabljen čas
Node2vec	4,5 GB	80 s
Hevristični pristop	110 MB	54 s

Tabela 4.1: Meritve hitrosti in porabe pomnilnika na naboru podatkov BlogCatalog

merjeni program	porabljen pomnilnik	porabljen čas
Node2vec	210 GB	380 min
Hevristični pristop	34 GB	545 min

Tabela 4.2: Meritve hitrosti in porabe pomnilnika na naboru podatkov LiveJournal

4.2 Primerjava časovne in prostorske zahtevnosti

Hitrost in poraba pomnilnika je bila testirana na naborih podatkov BlogCatalog in LiveJournal. Vse primerjave so bile narejene na računalniku s 144 jedrnim 2.50 GHz 64-bitnim procesorjem in 2TB pomnilnika. Poraba pomnilnika je bila merjena z ukazom smem. Omrežje v naboru podatkov BlogCatalog ima 10.312 vozlišč in 333.983 povezav. Omrežje v naboru podatkov LiveJournal ima 4.847.571 vozlišč in 68.993.773 povezav.

Kot je razvidno iz tabel 4.1 in 4.2, hevristični pristop porabi bistveno manj pomnilnika kot Node2vec. Hitrost izvajanja je primerljiva z algoritmom Node2vec, odvisna pa je tudi od samih podatkov. V omrežju BlogCatalog je bila povprečna stopnja vozlišča veliko višja kot v omrežju LiveJournal, zato je faktor razlike porabe pomnilnika med hevrističnim pristopom in algoritmom Node2vec veliko višji. V omrežju LiveJournal je bila povprečna stopnja vozlišča nižja, posledično se je poraba pomnilnika razlikovala le za faktor 6. Značilke v omrežju LiveJournal algoritem Node2vec izračuna hitreje kot hevristični pristop. To je posledica nizke povprečne stopnje vozlišča. Pred-

prirava verjetnostnih tabel namreč ni tako obširna, simuliranje naključnih sprehodov pa je pri algoritmu Node2vec očitno hitrejše kot pri hevrističnem pristopu.

Poglavje 5

Zaključek

Glavni namen tega diplomskega dela je bil zmanjšati pomnilniško zahtevnost algoritma Node2vec. Razvit je bil hevristični pristop za simuliranje naključnih sprehodov, ki imitira obnašanje algoritma Node2vec, a ima nižjo pomnilniško zahtevnost.

Razviti hevristični pristop k simuliranju naključnih sprehodov izračuna enako kvalitetne značilke kot algoritem Node2vec. Pri tem porabi veliko manj pomnilnika in zagotavlja linearnost pomnilniške in časovne zahtevnosti tudi v zelo gostih omrežjih. Hevristični pristop sicer zmanjša tudi časovno zahtevnost simuliranja naključnih sprehodov, a se na naborih podatkov v praksi to ne pozna bistveno. Na povprečnem domačem računalniku lahko torej izračunamo značilke omrežij z do 10^6 vozlišči, kar opazno preseže zmogljivost algoritma Node2vec, ki je to mejo dosegel že na nekaterih omrežjih z 10^4 vozlišči.

Za manjšo porabo pomnilnika pri hevurističnem pristopu plačamo ceno. V nekaterih primerih, kjer je graf relativno redek, je računanje značilk počasnejše kot pri algoritmu Node2vec, kar pa si lahko privoščimo, saj značilke omrežja tipično računamo redko. Izgubimo tudi možnost večkratne iteracije čez simulirane sprehode, vendar to ne predstavlja ovire. Nove sprehode generiramo dovolj hitro, da je namesto ponovne uporabe bolj smiselno generirati več novih sprehodov. Vhodna parametra p in q sta dobila drugačen pomen.

Kljub temu, da se hevristični pristop obnaša podobno kot Node2vec, je postopek izbire naslednjega vozlišča manj intuitiven, verjetnostna porazdelitev pa neočitna.

V nadaljnjih raziskavah bi lahko poskusili izboljšati računanje značilk. Postopek za računanje značilk je namreč še vedno zelo podoben algoritmu Word2vec in ne izkorišča dejstva, da računamo značilke vozlišč omrežja in ne značilk besed. Kljub temu, da računanje značilk pomnilniško ni zelo zahtevno, še vedno predstavlja časovno najzahtevnejši del algoritma. V nadaljnjih raziskavah bi lahko poiskali druge algoritme, ki podobno uporabljajo naključne sprehode drugega reda in bi jim s podobnim pristopom lahko znižali pomnilniško zahtevnost.

Literatura

[1] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. Group Formation in large Social Networks: Membership, Growth and Evolution. Dosegljivo: `https://snap.stanford.edu/data/soc-LiveJournal1.html`, 2006.

[2] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[3] B.-J. Breitkreutz, C. Start, T. Reguly, L. Boucher, A. Breitkreuz, M. Livstone, R. Oughtred, D. H. Lackner, J. Bähler, and V. et al. Wood. The BioGRID interaction database. Nucleic acids research, 36:D637-D640, 2008.

[4] Thomas H. Cormen, Charles E. Leiserson, L Rivest, Ronald, and Clifford Stein. *Introduction to Algorithms, third edition.* MIT, 2009.

[5] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *ACM SIGKIDD International Conference on Knowledge Discovery and Data Mining (KDD).* ACM, 2016.

[6] Stanford University Infolab Laboratory. Stanford Network Analysis Platform. Dosegljivo: `https://github.com/snap-stanford/snap`, 2016.

[7] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Conference on Neural Information Processing Systems.* ACM, 2013.

[8] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Conference on Neural Information Processing Systems*. ACM, 2013.

[9] Bryan Perrozi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representation. In *ACM SIGKIDD International Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 2014.

[10] Michael D. Vose. A linear algorithm for generating random numbers with a given distribution. *IEEE Transactions on Software Engineering*, 17(9):972–975, 1991.

[11] R. Zafarani and Liu H. Social computing data repository at ASU, 2009.